

System *F* with exceptions

Sylvain Lebresne

Logical, INRIA and PPS, Université Paris 7

December 13, 2007 – Tallinn, Estonia

1 Introduction

2 Presenting F_X

- The terms side
- The types side

3 A realisability model of F_X

4 Conclusion and perspectives

What are exceptions ?

- Exceptions are a convenient mechanism to handle “errors” or “exceptional behavior”.

What are exceptions ?

- Exceptions are a convenient mechanism to handle “errors” or “exceptional behavior”.
- It is used in many (most) programming languages : OCaml, Java, Ruby,

What are exceptions ?

- Exceptions are a convenient mechanism to handle “errors” or “exceptional behavior”.
- It is used in many (most) programming languages : OCaml, Java, Ruby,
- In functional languages :
 - Mostly associated to call-by-value reduction.

What are exceptions ?

- Exceptions are a convenient mechanism to handle “errors” or “exceptional behavior”.
- It is used in many (most) programming languages : OCaml, Java, Ruby,
- In functional languages :
 - Mostly associated to call-by-value reduction.
 - Generally not precisely typed.

What are exceptions ?

- Exceptions are a convenient mechanism to handle “errors” or “exceptional behavior”.
- It is used in many (most) programming languages : OCaml, Java, Ruby,
- In functional languages :
 - Mostly associated to call-by-value reduction.
 - Generally not precisely typed.
- What about exceptions in type theoretical settings?

Exceptions propagation

An exception usually interrupts all computations until reaching an appropriate handler:

Exceptions propagation

An exception usually interrupts all computations until reaching an appropriate handler:

$$\blacksquare (\text{raise } E) A \succ \text{raise } E \quad (\text{raise}_{cbn})$$

Exceptions propagation

An exception usually interrupts all computations until reaching an appropriate handler:

- $(\text{raise } E) A \succ \text{raise } E$ (raise_{cbn})
- $F (\text{raise } E) \succ \text{raise } E$ (raise_{cbv})

Exceptions propagation

An exception usually interrupts all computations until reaching an appropriate handler:

$$\blacksquare (\text{raise } E) A \succ \text{raise } E \quad (\text{raise}_{cbn})$$

$$\blacksquare F (\text{raise } E) \succ \text{raise } E \quad (\text{raise}_{cbv})$$

We do not want to keep the rule (raise_{cbv}) because:

Exceptions propagation

An exception usually interrupts all computations until reaching an appropriate handler:

$$\blacksquare (\text{raise } E) A \succ \text{raise } E \quad (\text{raise}_{cbn})$$

$$\blacksquare F (\text{raise } E) \succ \text{raise } E \quad (\text{raise}_{cbv})$$

We do not want to keep the rule (raise_{cbv}) because:

- it is more related to call-by-value β evaluation. Makes less sense with call-by-name.

Exceptions propagation

An exception usually interrupts all computations until reaching an appropriate handler:

$$\blacksquare (\text{raise } E) A \succ \text{raise } E \quad (\text{raise}_{cbn})$$

$$\blacksquare F (\text{raise } E) \succ \text{raise } E \quad (\text{raise}_{cbv})$$

We do not want to keep the rule (raise_{cbv}) because:

- it is more related to call-by-value β evaluation. Makes less sense with call-by-name.
- with (raise_{cbn}) , not confluent.

Exceptions as values

Exceptions are not anymore modifications of the control flow, but rather a special kind of values.

Exceptions as values

Exceptions are not anymore modifications of the control flow, but rather a special kind of values.

In particular, some new values appear. For example, if we use primitive natural numbers with 0 and S , “ S (raise E)” is not reducible.

Exceptions as values

Exceptions are not anymore modifications of the control flow, but rather a special kind of values.

In particular, some new values appear. For example, if we use primitive natural numbers with 0 and S , “ S (raise E)” is not reducible.

And we want to be able to type such a value.

1 Introduction

2 Presenting F_X

- The terms side
- The types side

3 A realisability model of F_X

4 Conclusion and perspectives

What is F_X ?

An extension of System F_η with typed exceptions and primitive natural numbers.

What is F_X ?

An extension of System F_η with typed exceptions and primitive natural numbers.

- A subtyping relation noted \leq .

What is F_X ?

An extension of System F_η with typed exceptions and primitive natural numbers.

- A subtyping relation noted \leq .
- With raise and try constructions.

What is F_X ?

An extension of System F_η with typed exceptions and primitive natural numbers.

- A subtyping relation noted \leq .
- With `raise` and `try` constructions.
- With new type constructions to type it.

What is F_X ?

An extension of System F_η with typed exceptions and primitive natural numbers.

- A subtyping relation noted \leq .
- With `raise` and `try` constructions.
- With new type constructions to type it.
- All the typing and subtyping rules of F_η are unchanged. We only add rules.

What is F_X ?

An extension of System F_η with typed exceptions and primitive natural numbers.

- A subtyping relation noted \leq .
- With `raise` and `try` constructions.
- With new type constructions to type it.
- All the typing and subtyping rules of F_η are unchanged. We only add rules.
- Justified by a realisability model.

F_X syntax

\mathcal{E} : a countable set of names of exceptions.

F_X syntax

\mathcal{E} : a countable set of names of exceptions.

The syntax of *F_X* is defined by:

$$M, N ::=$$

F_X syntax

\mathcal{E} : a countable set of names of exceptions.

The syntax of *F_X* is defined by:

$$M, N ::= x \mid \lambda x. M \mid M N$$

F_X syntax

\mathcal{E} : a countable set of names of exceptions.

The syntax of *F_X* is defined by:

$$M, N ::= x \mid \lambda x. M \mid M N \mid \text{raise } \varepsilon \mid \text{try } M \text{ with } \varepsilon \mapsto N$$

where $\varepsilon \in \mathcal{E}$.

F_X syntax

\mathcal{E} : a countable set of names of exceptions.

The syntax of *F_X* is defined by:

$$M, N ::= x \mid \lambda x. M \mid M N \mid \text{raise } \varepsilon \mid \text{try } M \text{ with } \varepsilon \mapsto N \\ \mid 0 \mid S \mid \text{Rec} \mid \text{eval}$$

where $\varepsilon \in \mathcal{E}$.

F_X syntax

\mathcal{E} : a countable set of names of exceptions.

The syntax of *F_X* is defined by:

$$M, N ::= x \mid \lambda x. M \mid M N \mid \text{raise } \varepsilon \mid \text{try } M \text{ with } \varepsilon \mapsto N \\ \mid 0 \mid S \mid \text{Rec} \mid \text{eval}$$

where $\varepsilon \in \mathcal{E}$.

We also define a notion of value:

$$V ::= \lambda x. M \mid 0 \mid S \mid S N \mid \text{Rec} \mid \text{Rec } M \mid \text{Rec } M N \mid \text{eval}$$

Reduction in F_X (1/2)

The notion of reduction is defined by the following rules:

Reduction in *F_X* (1/2)

The notion of reduction is defined by the following rules:

$$(\lambda x. M) N \quad > \quad M\{x := N\}$$

Reduction in *F_X* (1/2)

The notion of reduction is defined by the following rules:

$$\begin{array}{lcl}
 (\lambda x. M) N & > & M\{x := N\} \\
 (\text{raise } \varepsilon) M & > & \text{raise } \varepsilon \\
 \text{try } (\text{raise } \varepsilon) \text{ with } \varepsilon \mapsto N & > & N \\
 \text{try } (\text{raise } \varepsilon') \text{ with } \varepsilon \mapsto N & > & \text{raise } \varepsilon' \\
 \text{try } V \text{ with } \varepsilon \mapsto N & > & V
 \end{array}$$

Reduction in F_X (1/2)

The notion of reduction is defined by the following rules:

$(\lambda x. M) N$	$>$	$M\{x := N\}$
$(\text{raise } \varepsilon) M$	$>$	$\text{raise } \varepsilon$
$\text{try}(\text{raise } \varepsilon)\text{with } \varepsilon \mapsto N$	$>$	N
$\text{try}(\text{raise } \varepsilon')\text{with } \varepsilon \mapsto N$	$>$	$\text{raise } \varepsilon'$
$\text{try } V \text{with } \varepsilon \mapsto N$	$>$	V
$\text{Rec } X \ Y \ 0$	$>$	X
$\text{Rec } X \ Y \ (S \ N)$	$>$	$Y \ N \ (\text{Rec } X \ Y \ N)$
$\text{Rec } X \ Y \ (\text{raise } \varepsilon)$	$>$	$\text{raise } \varepsilon$

Reduction in F_X (2/2)

eval : “to bring exceptions up”

Reduction in F_X (2/2)

eval : “to bring exceptions up”

- $\text{eval } S (S (S (S (\text{raise } \varepsilon)))) > \text{raise } \varepsilon$

Reduction in F_X (2/2)

eval : “to bring exceptions up”

- $\text{eval } S (S (S (S (\text{raise } \varepsilon)))) > \text{raise } \varepsilon$
- $\text{eval } S (S (S (S 0))) > S (S (S (S 0)))$

Reduction in *F_X* (2/2)

eval : “to bring exceptions up”

- $\text{eval } S (S (S (S (\text{raise } \varepsilon)))) > \text{raise } \varepsilon$
- $\text{eval } S (S (S (S 0))) > S (S (S (S 0)))$

Easily defined with Rec and by using an accumulator.

Types of *F_x*

The syntax for the types of *F_x* is given by:

$$A, B ::=$$

Types of *F_x*

The syntax for the types of *F_x* is given by:

$$A, B ::= \alpha \mid \mathbb{N} \mid A \rightarrow B \mid \forall \alpha. A$$

Types of *F_X*

The syntax for the types of *F_X* is given by:

$$A, B ::= \alpha \mid \mathbb{N} \mid A \rightarrow B \mid \forall \alpha. A \\ \mid A \star \Delta \quad (\text{terms from } A \text{ or exceptions of } \Delta)$$

Où $\Delta \subseteq \mathcal{E}$

Types of *F_X*

The syntax for the types of *F_X* is given by:

$$\begin{aligned}
 A, B ::= & \alpha \mid \mathbb{N} \mid A \rightarrow B \mid \forall \alpha. A \\
 & \mid A \uplus \Delta \quad (\text{terms from } A \text{ or exceptions of } \Delta) \\
 & \mid A^\Delta \quad (\text{terms corrupted by exceptions of } \Delta)
 \end{aligned}$$

Où $\Delta \subseteq \mathcal{E}$



$A \uplus \Delta$: terms of type A or exceptions of names in Δ .



$A \cup \Delta$: terms of type A or exceptions of names in Δ .

- $0 : \mathbb{N} \cup \{\varepsilon\}$



$A \uplus \Delta$: terms of type A or exceptions of names in Δ .

- $0 : \mathbb{N} \uplus \{\varepsilon\}$
- $\text{raise } \varepsilon : \mathbb{N} \uplus \{\varepsilon\}$



$A \uplus \Delta$: terms of type A or exceptions of names in Δ .

- $0 : \mathbb{N} \uplus \{\varepsilon\}$
- `raise ε` : $\mathbb{N} \uplus \{\varepsilon\}$

Terms of $A \uplus \Delta$ are the ones caught by a try.

but it has its limitations

We still have some problems unsolved.

but it has its limitations

We still have some problems unsolved.

What type to give to S (raise ε) ?

but it has its limitations

We still have some problems unsolved.

What type to give to S (raise ε) ?

The type $\mathbb{N} \star \{\varepsilon\}$ is of course not appropriate.

but it has its limitations

What about $(A \rightarrow B) \uplus \Delta$?

but it has its limitations

What about $(A \rightarrow B) \uplus \Delta$?

$$f : (A \rightarrow B) \uplus \{\varepsilon\}$$

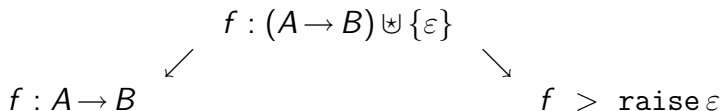
but it has its limitations

What about $(A \rightarrow B) \uplus \Delta$?

$$f : A \rightarrow B \quad \swarrow \quad f : (A \rightarrow B) \uplus \{\varepsilon\}$$

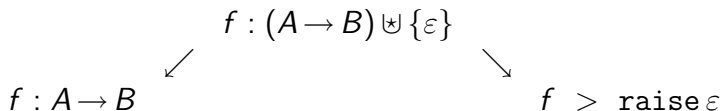
but it has its limitations

What about $(A \rightarrow B) \uplus \Delta$?



but it has its limitations

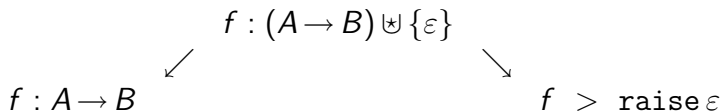
What about $(A \rightarrow B) \uplus \Delta$?



But f is not allowed by typing to be applied to a term of A

but it has its limitations

What about $(A \rightarrow B) \uplus \Delta$?



But f is not allowed by typing to be applied to a term of A
 Type construction $A \uplus \Delta$ doesn't deal well with arrow types.

And so comes the corruption

A term of A^Δ is a term of A for which some subterms could have been replaced by an exception.

And so comes the corruption

A term of A^Δ is a term of A for which some subterms could have been replaced by an exception.

Another way to see it,

And so comes the corruption

A term of A^Δ is a term of A for which some subterms could have been replaced by an exception.

Another way to see it,

- a term of $A \uplus \Delta$ may raise an exception at top-level.

And so comes the corruption

A term of A^Δ is a term of A for which some subterms could have been replaced by an exception.

Another way to see it,

- a term of $A \uplus \Delta$ may raise an exception at top-level.
- a term of A^Δ may raise an exception in some evaluation context (of A).

And so comes the corruption

A term of A^Δ is a term of A for which some subterms could have been replaced by an exception.

Another way to see it,

- a term of $A \uplus \Delta$ may raise an exception at top-level.
- a term of A^Δ may raise an exception in some evaluation context (of A).

Remark: A^Δ is a broader type than $A \uplus \Delta$.

Corruption main property

The main property that corruption enjoys is

Corruption main property

The main property that corruption enjoys is

$$(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta$$

Corruption main property

The main property that corruption enjoys is

$$(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta$$

with that we have:

$$S : \mathbb{N} \rightarrow \mathbb{N}$$

Corruption main property

The main property that corruption enjoys is

$$(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta$$

with that we have:

$$S : \mathbb{N} \rightarrow \mathbb{N} \leq (\mathbb{N} \rightarrow \mathbb{N}) \uplus \Delta$$

Corruption main property

The main property that corruption enjoys is

$$(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta$$

with that we have:

$$S : \mathbb{N} \rightarrow \mathbb{N} \leq (\mathbb{N} \rightarrow \mathbb{N}) \uplus \Delta \leq (\mathbb{N} \rightarrow \mathbb{N})^\Delta$$

Corruption main property

The main property that corruption enjoys is

$$(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta$$

with that we have:

$$S : \mathbb{N} \rightarrow \mathbb{N} \leq (\mathbb{N} \rightarrow \mathbb{N}) \uplus \Delta \leq (\mathbb{N} \rightarrow \mathbb{N})^\Delta = \mathbb{N}^\Delta \rightarrow \mathbb{N}^\Delta$$

Corruption main property

The main property that corruption enjoys is

$$(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta$$

with that we have:

$$S : \mathbb{N} \rightarrow \mathbb{N} \leq (\mathbb{N} \rightarrow \mathbb{N}) \uplus \Delta \leq (\mathbb{N} \rightarrow \mathbb{N})^\Delta = \mathbb{N}^\Delta \rightarrow \mathbb{N}^\Delta$$

and finally:

$$S(\text{raise } \varepsilon) : \mathbb{N}^{\{\varepsilon\}}$$

Typing rules of *F_X*

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \text{ (ax)}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \text{ (abs)}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \text{ (app)}$$

Typing rules of *F_X*

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \text{ (ax)}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \text{ (abs)}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \text{ (app)}$$

$$\frac{\Gamma \vdash M : A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash M : \forall \alpha. A} \text{ (gen)}$$

Typing rules of *F_X*

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \text{ (ax)}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \text{ (abs)}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \text{ (app)}$$

$$\frac{\Gamma \vdash M : A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash M : \forall \alpha. A} \text{ (gen)}$$

$$\frac{\Gamma \vdash M : A \quad A \leq B}{\Gamma \vdash M : B} \text{ (subs)}$$

Typing rules of F_X

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \text{ (ax)}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \text{ (abs)} \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \text{ (app)}$$

$$\frac{\Gamma \vdash M : A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash M : \forall \alpha. A} \text{ (gen)} \quad \frac{\Gamma \vdash M : A \quad A \leq B}{\Gamma \vdash M : B} \text{ (subs)}$$

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}} \text{ (zero)} \quad \frac{}{\Gamma \vdash S : \mathbb{N} \rightarrow \mathbb{N}} \text{ (succ)}$$

$$\frac{}{\Gamma \vdash \text{Rec} : \forall \alpha. \alpha \rightarrow (\mathbb{N} \rightarrow \alpha \rightarrow \alpha) \rightarrow \mathbb{N} \rightarrow \alpha} \text{ (rec)}$$

Typing rules of F_X continued

$$\frac{}{\Gamma \vdash \text{raise } \varepsilon : \emptyset \star \{\varepsilon\}} \text{ (raise)} \quad \emptyset \equiv \forall \alpha. \alpha$$

Typing rules of F_X continued

$$\overline{\Gamma \vdash \text{raise } \varepsilon : \emptyset \star \{\varepsilon\}} \quad (\text{raise}) \quad \emptyset \equiv \forall \alpha. \alpha$$

$$\frac{\Gamma \vdash M : A \star \{\varepsilon\} \quad \Gamma \vdash N : A}{\Gamma \vdash \text{try } M \text{ with } \varepsilon \mapsto N : A} \quad (\text{try})$$

Typing rules of F_X continued

$$\frac{}{\Gamma \vdash \text{raise } \varepsilon : \emptyset \uplus \{\varepsilon\}} \text{ (raise)} \quad \emptyset \equiv \forall \alpha. \alpha$$

$$\frac{\Gamma \vdash M : A \uplus \{\varepsilon\} \quad \Gamma \vdash N : A}{\Gamma \vdash \text{try } M \text{ with } \varepsilon \mapsto N : A} \text{ (try)}$$

$$\frac{}{\Gamma \vdash \text{eval} : \mathbb{N}^\Delta \rightarrow \mathbb{N} \uplus \Delta} \text{ (eval)}$$

First subtyping rules: F_{η}

$$\frac{}{A \leq A} \text{ (st-id)} \quad \frac{A \leq B \quad B \leq C}{A \leq C} \text{ (st-trans)}$$

First subtyping rules: F_{η}

$$\frac{}{A \leq A} \text{ (st-id)} \quad \frac{A \leq B \quad B \leq C}{A \leq C} \text{ (st-trans)}$$

$$\frac{A' \leq A \quad B \leq B'}{A \rightarrow B \leq A' \rightarrow B'} \text{ (st-arrow)}$$

First subtyping rules: F_{η}

$$\frac{}{A \leq A} \text{ (st-id)} \quad \frac{A \leq B \quad B \leq C}{A \leq C} \text{ (st-trans)}$$

$$\frac{A' \leq A \quad B \leq B'}{A \rightarrow B \leq A' \rightarrow B'} \text{ (st-arrow)}$$

$$\frac{A \leq B \quad \alpha \notin FV(A)}{A \leq \forall \alpha. B} \text{ (f-gen)} \quad \frac{}{\forall \alpha. A \leq A\{\alpha := B\}} \text{ (f-inst)}$$

$$\frac{\alpha \notin FV(A)}{\forall \alpha. (A \rightarrow B) \leq A \rightarrow \forall \alpha. B} \text{ (f-arr)}$$

Subtyping rules: $A \sqcup \Delta$

$$\frac{}{A \leq A \sqcup \Delta} \text{ (ex-uni)}$$

Subtyping rules: $A \uplus \Delta$

$$\frac{}{A \leq A \uplus \Delta} \text{ (ex-uni)}$$

$$\frac{}{A \uplus \emptyset \leq A} \text{ (ex-noex)}$$

Subtyping rules: $A \sqcup \Delta$

$$\frac{}{A \leq A \sqcup \Delta} \text{ (ex-uni)} \qquad \frac{}{A \sqcup \emptyset \leq A} \text{ (ex-noex)}$$

$$\frac{A \leq B}{A \sqcup \Delta \leq B \sqcup \Delta} \text{ (ex-ctx)}$$

Subtyping rules: $A \uplus \Delta$

$$\frac{}{A \leq A \uplus \Delta} \text{ (ex-uni)} \quad \frac{}{A \uplus \emptyset \leq A} \text{ (ex-noex)}$$

$$\frac{A \leq B}{A \uplus \Delta \leq B \uplus \Delta} \text{ (ex-ctx)}$$

$$\frac{}{\forall \alpha. (A \uplus \Delta) \leq (\forall \alpha. A) \uplus \Delta} \text{ (ex-fallu)}$$

$$\frac{}{(A \uplus \Delta) \uplus \Delta' = A \uplus (\Delta \cup \Delta')} \text{ (eq-uu)}$$

Subtyping rules: A^Δ

$$\frac{}{A \Downarrow \Delta \leq A^\Delta} \text{ (ex-corrupt)}$$

Subtyping rules: A^Δ

$$\frac{}{A \uplus \Delta \leq A^\Delta} \text{ (ex-corrupt)}$$

$$\frac{}{\emptyset^\Delta \leq \emptyset \uplus \Delta} \text{ (ex-empty)}$$

Subtyping rules: A^Δ

$$\frac{}{A \uplus \Delta \leq A^\Delta} \text{ (ex-corrupt)} \qquad \frac{}{\emptyset^\Delta \leq \emptyset \uplus \Delta} \text{ (ex-empty)}$$

$$\frac{}{\forall \alpha. A^\Delta \leq (\forall \alpha. A)^\Delta} \text{ (ex-fallc)}$$

Subtyping rules: A^Δ

$$\frac{}{A \uplus \Delta \leq A^\Delta} \text{ (ex-corrupt)} \quad \frac{}{\emptyset^\Delta \leq \emptyset \uplus \Delta} \text{ (ex-empty)}$$

$$\frac{}{\forall \alpha. A^\Delta \leq (\forall \alpha. A)^\Delta} \text{ (ex-fallc)}$$

$$\frac{}{(A^\Delta)^{\Delta'} = A^{(\Delta \cup \Delta')}} \text{ (eq-cc)} \quad \frac{}{(A \uplus \Delta)^{\Delta'} = A^{\Delta'} \uplus \Delta} \text{ (eq-uc)}$$

Subtyping rules: A^Δ

$$\frac{}{A \uplus \Delta \leq A^\Delta} \text{ (ex-corrupt)} \quad \frac{}{\emptyset^\Delta \leq \emptyset \uplus \Delta} \text{ (ex-empty)}$$

$$\frac{}{\forall \alpha. A^\Delta \leq (\forall \alpha. A)^\Delta} \text{ (ex-fallc)}$$

$$\frac{}{(A^\Delta)^{\Delta'} = A^{(\Delta \cup \Delta')}} \text{ (eq-cc)} \quad \frac{}{(A \uplus \Delta)^{\Delta'} = A^{\Delta'} \uplus \Delta} \text{ (eq-uc)}$$

$$\frac{}{(A \rightarrow B)^\Delta = A^\Delta \rightarrow B^\Delta} \text{ (eq-arrc)}$$

Examples

$$\text{pred} \equiv \text{Rec} (\text{raise } \varepsilon) (\lambda x. \lambda y. x) \quad :$$

Examples

$$\text{pred} \equiv \text{Rec} (\text{raise } \varepsilon) (\lambda x. \lambda y. x) \quad : \quad \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\}$$

Examples

$$\begin{aligned} \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) &: \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\ \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 &: \mathbb{N} \rightarrow \mathbb{N} \end{aligned}$$

Examples

$$\begin{aligned} \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) &: \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\ \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 &: \mathbb{N} \rightarrow \mathbb{N} \\ \text{add2} &\equiv \lambda n. S (S n) \end{aligned}$$

Examples

$$\begin{aligned}
 \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) &: & \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\
 \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 &: & \mathbb{N} \rightarrow \mathbb{N} \\
 \text{add2} &\equiv \lambda n. S (S n)
 \end{aligned}$$

We can define second order list and then:

Examples

$$\begin{aligned} \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) &: \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\ \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 &: \mathbb{N} \rightarrow \mathbb{N} \\ \text{add2} &\equiv \lambda n. S (S n) \end{aligned}$$

We can define second order list and then:

$\text{map } (\lambda n. \text{add2 } (\text{pred } n)) [3; 0; 4]$

Examples

$$\begin{aligned} \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) &: \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\ \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 &: \mathbb{N} \rightarrow \mathbb{N} \\ \text{add2} &\equiv \lambda n. S (S n) \end{aligned}$$

We can define second order list and then:

$$\text{map } (\lambda n. \text{add2 } (\text{pred } n)) [3; 0; 4] > [4; S (S (\text{raise } \varepsilon)); 5]$$

Examples

$$\begin{aligned} \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) & : & \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\ \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 & : & \mathbb{N} \rightarrow \mathbb{N} \\ \text{add2} &\equiv \lambda n. S (S n) \end{aligned}$$

We can define second order list and then:

$$\text{map } (\lambda n. \text{add2 } (\text{pred } n)) [3; 0; 4] > [4; S (S (\text{raise } \varepsilon)); 5] : \text{list}(\mathbb{N}^{\{\varepsilon\}})$$

Examples

$$\begin{aligned} \text{pred} &\equiv \text{Rec } (\text{raise } \varepsilon) (\lambda x. \lambda y. x) &: \mathbb{N} \rightarrow \mathbb{N} \uplus \{\varepsilon\} \\ \text{pred}' &\equiv \lambda n. \text{try } (\text{pred } n) \text{ with } \varepsilon \mapsto 0 &: \mathbb{N} \rightarrow \mathbb{N} \\ \text{add2} &\equiv \lambda n. S (S n) \end{aligned}$$

We can define second order list and then:

$$\text{map } (\lambda n. \text{add2 } (\text{pred } n)) [3; 0; 4] > [4; S (S (\text{raise } \varepsilon)); 5] : \text{list}(\mathbb{N}^{\{\varepsilon\}})$$

$$\text{head } [4; S (S (\text{raise } \varepsilon)); 5] > 4$$

1 Introduction

2 Presenting F_X

- The terms side
- The types side

3 A realisability model of F_X

4 Conclusion and perspectives

A realisability model

We have developed a realisability model of Fx .

A realisability model

We have developed a realisability model of Fx .

It is designed using orthogonality techniques.

A realisability model

We have developed a realisability model of F_x .

It is designed using orthogonality techniques.

The interpretations of $A \uplus \Delta$ and A^Δ follow the idea that

- $A \uplus \Delta$ may raise exceptions at top-level.
- A^Δ may raise exceptions in some evaluation context.

Arrow types interpretation is not standard

With corruption, the arrow types is not the usual realisability one anymore.

Arrow types interpretation is not standard

With corruption, the arrow types is not the usual realisability one anymore.

If we denote by \xrightarrow{r} the realisability arrow, our arrow is understand as:

$$A \rightarrow B = \bigcap_{\Delta \subseteq \mathcal{E}} (A^{\Delta} \xrightarrow{r} B^{\Delta})$$

Model properties

Theorem (Model soundness)

The model is sound.

Model properties

Theorem (Model soundness)

The model is sound.

Corollary (Type safety)

If M is a term such that $\vdash M : \mathbb{N}$, then $M \rightsquigarrow^ S^n 0$.*

Model properties

Theorem (Model soundness)

The model is sound.

Corollary (Type safety)

If M is a term such that $\vdash M : \mathbb{N}$, then $M \rightsquigarrow^ S^n 0$.*

Corollary (Weak normalization)

Terms of F_x are weakly normalizing.

1 Introduction

2 Presenting F_X

- The terms side
- The types side

3 A realisability model of F_X

4 Conclusion and perspectives

Conclusion

- A mechanism of exceptions adapted to call-by-name evaluation and a way to precisely type it based on subtyping.

Conclusion

- A mechanism of exceptions adapted to call-by-name evaluation and a way to precisely type it based on subtyping.
- It should be adaptable to dependent types.

Conclusion

- A mechanism of exceptions adapted to call-by-name evaluation and a way to precisely type it based on subtyping.
- It should be adaptable to dependent types.
- That's all, thanks for your attention.